Proceedings of the 2002 Winter Simulation Conference (WSC 2002).
E. Yücesan, C.-H. Chen, J. L. Snowdon, and , J. M. Charnes, eds. December 2002. San Diego. ISBN 0-7803-7615-3

THE APPLICATION OF DISTRIBUTED SIMULATION IN TOMAS: REDESIGNING A COMPLEX TRANSPORTATION MODEL

Mark B. Duinkerken
Jaap A. Ottjes
Gabriel Lodewijks

Faculty of OCP, Department of Mechanical Engineering and Marine Technology
Delft University of Technology
Mekelweg 2
2628 CD Delft, THE NETHERLANDS

ABSTRACT

This paper describes the application of distributed discrete event simulation in the study of an automated container terminal. The new model was developed to continue the study of large and complex logistic systems. In a previous study, a stand-alone model of the terminal was used that included all the characteristics of container handling between the ships and the container stack. A new distributed simulation model was developed by decomposing the original model into a distributed structure of communicating, small sub models. It is shown that with relative little effort and hardly any programming overhead, a complex stand-alone model can be decomposed into small, easy to understand sub models. The new distributed structure improves the transparency and maintainability of the simulation model, while guaranteeing the original benefits of the stand-alone model and the required reproducibility of the experiments.

1 INTRODUCTION

1.1 General

Discrete event simulation has proven to be an effective tool to study the performance of logistic systems and to aid in their design. Today, the scale and complexity of these systems is growing tremendously. The study of large and complex logistic systems needs to be performed by more than one expert because of the amount of modeling work and the mostly limited time frame available for the study. Stand-alone simulation models are therefore becoming too limited to sufficiently support the design of these systems. For this reason distributed simulation models have been developed. The development of these models was supported by the need to combine separately developed simulation models and to extend the use of developed control algorithms from simulation environments to prototyping or even real production and transportation systems. The development of distributed simulation models was possible since most hardware today, like computers, is connected in networks and since most programming environments support communication between hardware on various levels.

Distributed simulation is presented as a tool to enable the effective study of large and complex logistic systems and to enable the application of control algorithms that were developed within a simulation environment in a real control systems. Today, worldwide efforts are made to develop a general concept for distributed modeling and simulation (Fujii et al. 1999). At this moment HLA is becoming the standard (Page and Smith 1998). However, HLA is relatively difficult and inflexible to use in educational and research environments. In those environments a simple, open, flexible and free tool like TOMAS is required (Veeke and Ottjes 2001). TOMAS is used in the study presented in this paper.

TOMAS is based on the process interaction approach for discrete event simulation (Zeigler et al. 2000, Ottjes et al. 2001). A more elaborate discussion on this simulation tool can be found in (Veeke and Ottjes 2000, Veeke and Ottjes 2001). Within TOMAS, new methods are developed to support distributed modeling and simulation. Communication is established with string based messaging using sockets. Due to the open structure of TOMAS, a minimum programming effort is required to restructure a stand-alone model to a distributed set of models.

Running a distributed simulation however is not the same as parallel computing (Fujimoto 2001). Although a distributed simulation model can be run on more than one processor and therefore be faster, the main purpose is not to gain computational speed but to enable the study of large and complex logistic systems. In a stand-alone simulation model only one process can be current. In a distributed simulation only one member model can be active. The conservative synchronization of time guarantees the reproducibility of simulation results in both a stand-alone and a distributed version of the same model. In general, communication between hardware slows down the simulation speed. Therefore, the option to switch between the stand-alone and the distributed mode, back and forth, at any phase of a project is a major advantage. This aspect is also discussed in this paper.

1.2 Approach

Section 2 of this paper describes the development of distributed simulation in the discrete event simulation package TOMAS. Section 2 in particular describes the application of distributed simulation models, their conservative synchronization of time, and the communication aspects within the model.

The equipment used on a typical container terminal is described in section 3. A stand-alone model of this system is used for the study of automated container terminals. More information on this study can be found in (Duinkerken et al. 1999, Duinkerken and Ottjes 2000). This stand-alone model is used as a test case for developing a distributed simulation model, described in section 4. Following a few relative easy steps, the stand-alone model is split up into five member models. Each member model has its own controller that is responsible for the communication between the models. The communication from and to a member model is centralized in the controller. The interface of each member model therefore needs to be clearly defined. This allows easier development of alternative models with the same functionality.

Finally, section 5 describes the result of the study, the conclusions that can be drawn from them, and some future developments.

2 DISTRIBUTED SIMULATION MODELS

2.1 Application of Distributed Simulation Models

Today, simulation is used in the design of complex systems, including the design of logistic systems. Several developments changed the demands on the traditional simulation approach.

Firstly the size of the projects, in which simulation is being used, increased. A complex simulation study with limited time span is too large to be performed by only one expert. The ability to do concurrent engineering is required, so that more experts could cooperate in parallel. This demands a different model architecture.

Secondly the complexity of the design problems increased, which led to a design approach where simulation is needed during different design phases. A model cannot be specified immediately in all of its details. Some kind of aggregation facility is needed.

Thirdly the investments in programming efforts increased dramatically because of the growing degree of automation in real systems. There is a need to reuse the planning and control programming code from simulation models in the real environment.

Finally, because of the common use of simulation, it became a real topic to reuse some of the elements developed in earlier design projects. Object orientation has proven to be a real answer to this last aspect.

2.2 Conservative Synchronization

The synchronization to a common time-axis is essential for distributed simulation. Often the term "parallel simulation" is used, but "parallel" is more than just distributed. This paper is restricted to discrete process simulation as an expansion of discrete event simulation (Veeke and Ottjes 2000). A process is defined as an ordered sequence of time intervals that connect all the events of one element (class). This means that all state changes are performed at discrete moments in time and at any moment there is one and only one process "current".

If only one element can be current at one moment, in a distributed model only one member model can be current at any moment or even better: only one element of any member model can be current. The following rules are used:

- The first event generated for a specific moment will be the first event handled at that moment
- In case of state events, the state of the system will be checked after each event and not only when simulation time proceeds
- In a distributed model environment the member model with the smallest event time receives control. If two member models have the same event time, the member model with the event first generated will receive control, unless specifically requested by the modeler.

Note the word "generated" instead of "scheduled". The moment of event-generation is essential in preserving the reproducibility of the simulation runs. From the above can be concluded that a distributed model synchronizes events to a common time-axis by sequencing the first events of member models. Member models are assumed to sequence their events to their own local time-axis.

According to these rules distributed simulation still is quasi-parallel instead of parallel, because nothing is running in parallel; so distributed simulation in this sense doesn't result in faster models (as a consequence of distributing the member models over different PC's).

Finally distributed simulation should not be confused with real-time simulation. These definitions lead in fact to the conclusion that a combination of distributed and parallel simulation is needed to make it real-time.

2.3 Communication

As discussed in the previous paragraph, the events of all member models must be scheduled by one central server, the so-called timeserver. The timeserver receives all event-related messages from the member models, schedules the events in the correct sequence, and grants control to the first-scheduled member model. Beside the event-related messages, distribution leads to model-to-model communication. Because all member models are known to the timeserver, the timeserver will also act as a central "post-office" for all messages. The member models do not need to be aware of the location of all other member models, it is sufficient that each member model will connect to the timeserver.

3 THE AUTOMATED CONTAINER TERMINAL

3.1 Case Description

Figure 1 shows a typical container terminal. Containers are stacked in a large stacking area. Fully automated stacking cranes service this stack. The transport of containers is provided by Automated Guided Vehicles (AGVs), using fixed routes. Quay cranes load and unload the container vessels.

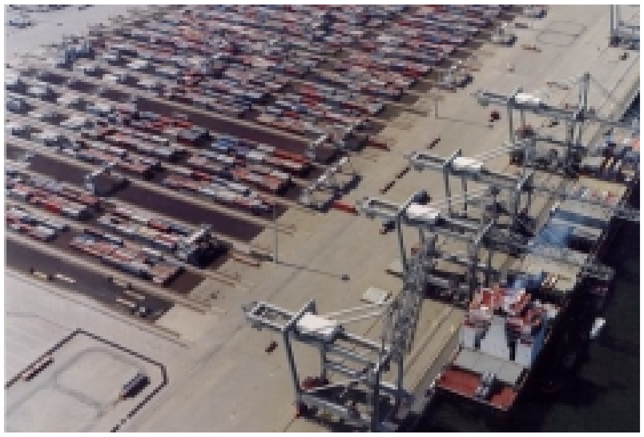


Figure 1 Overview of a Container Terminal

A simulation model for this system has been developed (Duinkerken et al. 1999, Duinkerken and Ottjes 2000) using the process interaction method. The main objects in our simulation model correspond with the physical objects: containers and containerships, quay cranes, AGVs and route-layout, stack and stacking cranes. One of the most important elements of our model however is virtual, namely the control system, the planning and control module. The control system is responsible for the scheduling of all operations on the terminal. The elements within this container terminal are discussed in more detail in the following paragraphs.

3.2 Stack System

The stack system consists of a number of stacking lanes. A stacking lane is specified by its width and length. Each stacking lane uses its own stacking crane. The stacking lane is connected with the quay infrastructure by 4 AGV transfer points per stacking lane. Each stacking lane is divided in an export-area and an import-area. All containers for loading will be placed in the export-area; all containers that are unloaded from a ship are placed in the import-area. The physical sizes of stacking lanes are user-defined.

Each stacking lane is equipped with one Automated Stacking Crane (ASC). The stacking crane provides for both the stacking of incoming load containers and the removal of outgoing ones. The stacking has priority, because this move will free an AGV. The sequencing of jobs is determined by the control system.

The performance of the stack is called the stack response time. This is the average time the stack will handle a container request. The performance can be subdivided in average move time inbound and average move time outbound.

3.3 Quay System

The quay system consists of quay cranes along the quay where ships can dock. The quay cranes stand in fixed positions on the quay so the traveling of the quay cranes along the quay is not simulated. This leads to a fixed length of the maximum waiting queue for the cranes. Quay cranes are loading and unloading AGVs. Most important during loading is the sequence in which the containers arrive. Usually, a ship is loaded with a strict loading sequence.

The performance of the quay system is characterized by the quay crane utilization. This is defined as the percentage of time that the quay crane is active, and thus complementary to the time the quay crane is waiting for AGVs. Because quay cranes are the most expensive equipment on the terminal, the objective of the control system is maximization of the quay crane utilization.

3.4 Transport System

The purpose of the quay transport system is to transport unloaded containers from the ship to the stack, and to bring load containers from the stack to the quay cranes. The AGVs are driving along fixed paths. The layout of the terminal in Figure 1 is shown in Figure 2. It includes 7 quay-cranes, 32 automatic stacking cranes, a stack for empty containers, and the current routing of the AGVs. On this terminal a maximum of 50 AGVs is in operation.

The traffic between stack and quay follows a circular pattern. Typically, the vehicles travel along the entire length of the ship and turn back along the stack. In the quay area a fixed traffic lane is reserved for each quay crane. In the stack-area the traffic for two quay-cranes is combined to form a single traffic lane.

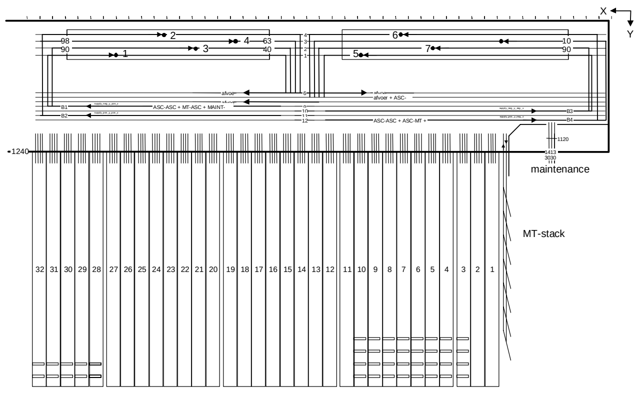


Figure 2 Layout of the DSL Terminal at the Maasvlakte Rotterdam

Quay transport uses Automated Guided Vehicles (AGVs) for the pick-up and delivery of containers at the quay cranes. The quay transport system is described comprehensively in (Duinkerken et al. 1999). In this work a method for the design of multi-AGV systems and control of their operation is presented.

3.5 Container System

The container system keeps track of all containers at the container terminal. Two types of containers are distinguished. Unload containers are containers on a ship that are unloaded by the quay crane and stacked in the import stack. Load containers are defined as the containers that start in the stack, and are loaded onto a ship. The model defines a shipload as a set of holds. Each hold defines a set of unload-containers and a set of load containers. For each hold the quay crane that will handle the loading and unloading is pre-defined.

3.6 Control System

The control system is responsible for the scheduling of all operations on the terminal. Main principle in the control system is to make decisions local instead of central, and as late as possible. The main driver of all scheduling is formed by the load plans of the container ships. Within a ship hold, the loading sequence for the load containers is determined by the load plan. This plan takes into account the destination of a container and various other properties including size, and weight. The possibility of relaxing the load plan by introducing "load categories" has been investigated. Instead of a fixed sequence of individual containers, the containers are grouped in categories, and a fixed sequence remains only between the categories. We expect that the resulting freedom of sequence within a category will improve both the quay transport and the stack response times.

4 IMPLEMENTATION

4.1 The Distributed Architecture

The simulation model of the container terminal, presented in the previous section, has evolved in the last years from a tiny simulation model to a complex system, modeling all quay-side terminal operations. Although this stand-alone model was originally not built to be decomposed for distribution, the member models can be distinguished. The description in the previous section strongly suggests an architecture of the distributed model as is given in Figure 3.

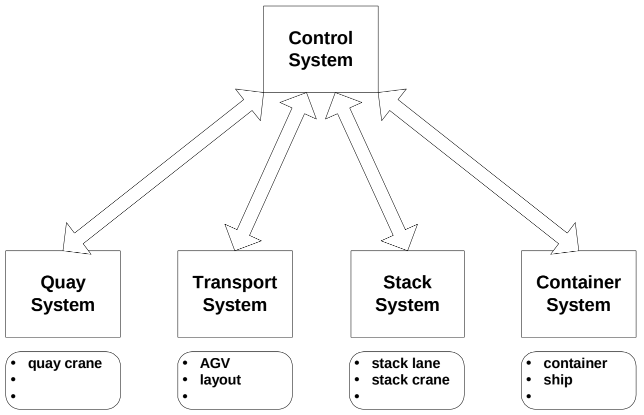


Figure 3 The Five Member Models with Some Important Object-Classes

While the architecture is obvious, the method to transform the model from standalone to distributed is not. The standalone model is object oriented. Each physical object in the system has its simulation counterpart. This results in a flat structure where every object can "see" other objects and access its methods and properties because they are in the same "memory-space".

The distributed model consists of separated member models in different memory-spaces. Within a member model, objects can still "see" and access each other, but for interaction with objects in other member models, communication must be defined. So by distributing a stand-alone model, a hierarchy is re-introduced.

4.2 Restructuring the Model

Most objects in the stand-alone model are defined in separate units. In case an object needs to access another object, the unit of that object must be "included" during compilation time. Thus by studying the include statements of each unit, the relationships between all objects of the stand-alone model can be found.

To obtain a distributed architecture the following steps were taken:

- Combine the units that form a member model and add a controller. Every unit of the member model may only include its controller unit. Only the controller-unit may include other controller units. Communication between objects of different member models will now only be possible through the controllers. Controllers can only access the properties and methods of other controllers
- Detach the member models one by one by compiling them as independent simulation models. The separated controller can now only send messages to the controllers of other member models. On the receiving side, these messages are translated to the original method-calls.

The major difficulty in restructuring the model was caused by the units used in more than one subsystem. For example, the stack system, the quay system and the transport system all used the object "container", which in the new architecture is only existing in the container system. This can only be solved by giving each system its own implementation of the object container, containing only the properties which are relevant for that system.

It can be concluded that the process interaction method, used for the original standalone model, resulted in a model which was easily transformed to a distributed architecture.

4.3 Communication

In the distributed architecture, the method calls from controllers are replaced by string based messages. On receiving a message from another member model, the received text string must result in the proper method call. Also, if a method call contains parameters, these parameters must be translated to a string by the sending model, and be interpreted by the receiving model. A special kind of parameter is the pointer to an object. It is natural that objects are known by name. Each member model contains a method that will translate the name of an object into a pointer to that object.

Implementing a controller for each member model reintroduces a hierarchy, which was lost by building the standalone model object oriented! However, there is a strong argument in favor of this hierarchy. It forces the model builder to define a clear communication set between subsystems. This set must contain all the needed method calls between the member models. By carefully studying the communication set, the model-to-model communication can be streamlined, and reduced to the minimum needed. Also, a clear communication set is in fact the interface-definition of a member model, which allows the development of alternative models with the same functionality.

The types of messages in model-to-model communication are:

- Send information
- Request for information. The sending model must wait for a reply message containing the answer string
- Process-related message. The process of the sending model requires an object in another model to cancel or resume its process. The receiving model updates its events at the timeserver, before the sending model continues.

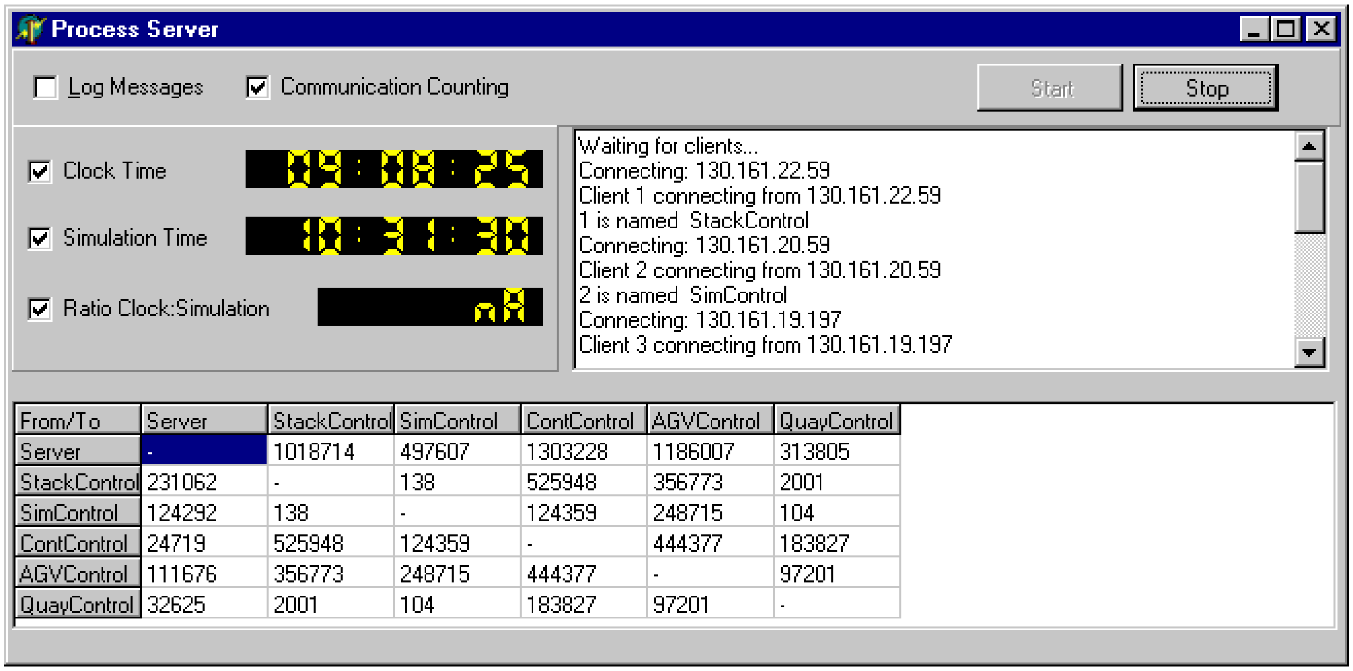
From the previous follows the need for synchronized communication. The model which sends a messages waits for confirmation before continuing its process. As mentioned before, the models will not run parallel. At each moment only one system can be current. However, this is done to guarantee the reproducibility of the simulation results.

5 RESULTS AND CONCLUSIONS

5.1 Results

The implementation of the distributed architecture using the proposed approach was successful. In Figure 4 a screen dump of the timeserver is presented. Because the timeserver is also responsible for the model-to-model communication, all the messages send during simulation can be counted. A table with the counted messages is available at the timeserver.

There are two kinds of communication: server-model and model-model. The server-model communication is



[ ] Log Messages [✓] Communication Counting

[✓] Clock Time 09:08:25

[✓] Simulation Time 10:31:30

[✓] Ratio Clock:Simulation nA

```
Waiting for clients...
Connecting: 130.161.22.59
Client 1 connecting from 130.161.22.59
1 is named  StackControl
Connecting: 130.161.20.59
Client 2 connecting from 130.161.20.59
2 is named  SimControl
Connecting: 130.161.19.197
Client 3 connecting from 130.161.19.197
```

From/To	Server	StackControl	SimControl	ContControl	AGVControl	QuayControl
Server	-	1018714	497607	1303228	1186007	313805
StackControl	231062	-	138	525948	356773	2001
SimControl	124292	138	-	124359	248715	104
ContControl	24719	525948	124359	-	444377	183827
AGVControl	111676	356773	248715	444377	-	97201
QuayControl	32625	2001	104	183827	97201	-

Figure 4 Screen Dump of the TOMAS Timeserver

related to the synchronization of the processes. The event-related messages are generated by the TOMAS environment and hidden for the user, but counted in the table. The effect of synchronized model-model communication can be seen in the figure: because each message is replied, this part of the matrix is symmetric.

5.2 Conclusions

It is shown that with relative little effort, the complex model could be decomposed into small, easy to understand member models. The new structure improves the transparency and maintainability of the simulation, while guaranteeing the original benefits of the model and the required reproducibility of the experiments. Because of the process interaction method for simulation, the standalone model was easily transformed to a distributed architecture. It is expected that in the future a reversed approach, building a complex model by implementing small member models, will be successful.

5.3 Future Developments

The main task for improvement of this model lies in reducing the size of the communication set, and the frequency of the messages. By logging the messages, the model builder gains insight in the amount of communication. Where in the standalone model accessing another object was convenient, in the distributed model it results in communication which slows down the simulation speed. Therefore, by making the communication more efficient, the total number of calls can be reduced significantly. This will result in higher simulation speed.

It is not uncommon that a simulation project starts relatively small, evolves and gets more complex over time. In fact, the terminal model presented here started as a study concerning AGV traffic on two crossing roads. When it is expected that a project will become complex, one should anticipate concurrent engineering by designing a distributed architecture. The main advantage of a stand-alone model is simulation speed. The ideal situation is therefore when the same project can be compiled both in stand-alone and distributed mode. The approach presented in this paper allows such mode-switching.

The next step is the development of a distributed simulation architecture to support the design of operations and control for container handling in a new part of the Port of Rotterdam. In this project, researchers from different faculties at the Delft University of Technology and the Erasmus University in Rotterdam cooperate. The architecture, called the "Backbone" will connect and synchronize several discrete simulation models during research and prototyping of different design phases. Common tasks for the support of a distributed simulation study like run control, scenario management, logging and animation, are implemented in the architecture itself.

Further work will include a laboratory of scaled AGVs (1:25) to cooperate within the architecture. At that moment, the paradigm of reproducibility must be relaxed, and parallel computing must be introduced, to allow real-time control of the AGVs.

REFERENCES

Duinkerken, M.B.; J.J.M. Evers; J.A. Ottjes. 1999. TRACES : Traffic Control Engineering System. Proceedings 31st Summer Computer Simulation Conference. Chicago [SCS] pp 461-465.

Duinkerken, M.B.; J.A. Ottjes. 2000. A simulation model for automated container terminals. Proceedings of the Business and Industry Simulation Symposium. Washington D.C. [SCS] pp 134-139.

Duinkerken, M.B.; J.J.M. Evers; J.A. Ottjes. 2001. A simulation model integrating quay transport and stacking policies on automated container terminals. Proceedings of the 15th European Simulation Multiconference. Prague [SCS] pp 909-916.

Fujii, S.; A. Ogita, Y. Kidani, T. Kaihara. 1999. Synchronization Mechanisms for Integration of Distributed Manufacturing Simulation Systems. Simulation 72:3, pp.187-197.

Fujimoto, R.M. 2001. Parallel and distributed simulation systems. Proceedings of the 2001 Winter Simulation Conference. Arlington, VA.

Ottjes, J.A.; H.P.M. Veeke; A.A. Buizer. 2001. Experimenting with distributed modeling and simulation using the internet. Proceedings of the 15th European Simulation Multiconference. Prague [SCS] pp 571-577.

Page, E. H.; R. Smith. 1998. Introduction to military training simulation: a guide for discrete event simulationists, Proceedings of the 1998 Winter Simulation Conference (WSC 98). Washington, DC pp. 53-60.

Veeke, H.P.M.; J.A. Ottjes. 2000. TOMAS: Tool for Object-Oriented Modelling and Simulation. Proceedings of the Business and Industry Simulation Symposium. Washington D.C. [SCS] pp 76-81.

Veeke, H.P.M.; J.A. Ottjes. 2001. Applied distributed discrete process simulation. Proceedings of the 15th European Simulation Multiconference. Prague [SCS] pp 641-648.

Zeigler, B.P.; H. Praehofer, T.G. Kim. 2000. Theory of Modeling and Simulation. 2 ed. Academic Press, New York.

AUTHOR BIOGRAPHIES

MARK B. DUINKERKEN obtained his Master degree in Applied Mathematics at the Delft University of Technology (1991). He is currently an Assistant Professor in Logistic Engineering at the faculty of Design, Engineering and Production, section Transport and Logistic Technology, of Delft University of Technology.

His specialization is modeling and simulation of logistic processes and optimizing the planning and control of logistic processes with OR-techniques. He participates in several public-private research projects aimed at the design of high capacity container terminals. Other work includes research concerning City Logistics, the development of an AGV-laboratory, student courses in computer simulation and the development of simulation tools. His e-mail address is `<m.b.duinkerken@wbmt.tudelft.nl>`.

JAAP A. OTTJES studied Physics at the Delft University of Technology and obtained his Master degree in 1970. He obtained his Ph.D. degree at Delft University of Technology on pneumatic transport. Jaap Ottjes is Associate Professor at the faculty of Design, Engineering and Production, section Transport and Logistic Technology, of Delft University of Technology.

He specialized in the logistic modeling and simulation of transportation and production systems. In this field, he worked as consultant on the modeling and control of the bottling process at Heineken. Currently, he is involved in several research projects concerning the design and modeling of automated harbors.

GABRIEL LODEWIJKS studied mechanical engineering at Twente University and Delft University of Technology, The Netherlands, from which he obtained a Master degree (cum laude) in 1992. He specialized in transport technology, material engineering and dynamics, and obtained his Ph.D. degree at Delft University of Technology on the dynamics of belt systems in 1996.

In November 2000 he was appointed Professor of Transport Technology and Logistic Technology at the faculty of Design, Engineering and Production, section Transport and Logistic Technology, of Delft University of Technology. His main interest is in belt conveyor technology, DEM applications in transport technology, automation of transport systems, material engineering and dynamics.